Insight from a Containerized Kubernetes Workload Introspection

Thomas Watts
The SSI Group, LLC
Thomas.Watts@ssigroup.com

Ryan Benton
The University of South Alabama
rbenton@southalabama.edu

David Bourrie
The University of South Alabama
dbourrie@southalabama.edu

Jordan Shropshire
Uversity of South Alabama
jshropshire@southalabama.edu

Abstract

Developments in virtual containers, especially in the cloud infrastructure, have led to diversification of jobs that containers are used to support, particularly in the big data and machine learning spaces. The diversification has been powered by the adoption of orchestration systems that marshal fleets of containers to accomplish complex programming tasks. The additional components in the vertical technology stack, plus the continued horizontal scaling have led to questions regarding how to forensically analyze complicated technology stacks. This paper proposed a solution through the use of introspection.

An exploratory case study has been conducted on a bare-metal cloud that utilizes Kubernetes, the introspection tool Prometheus, and Apache Spark. The contribution of this research is two-fold. First, it provides empirical support that introspection tools can acquire forensically viable data from different levels of a technology stack. Second, it provides the ground work for comparisons between different virtual container platforms.

1. Introduction

Virtual containers, which are an iteration of the concept of virtual machines, are frequently used in the cloud. Where a virtual machine is a complete operating system on top of virtualized hardware, a container is a lightweight silo for running an application on a host operating system [1]; they contain only the software needed to accomplish a specific task. They are used to accelerate processing in data processing frameworks such as Apache Spark [2]. Orchestrating containers through popular frameworks such as Kubernetes [3] or Apache YARN [4] has grown as a research area over the past several years.

Flexera's 2020 State of the Cloud survey [5] underscores how popular adoption of Kubernetes is with a ten percent adoption rate increase from forty-eight to fifty-eight percent from 2019 to 2020. That is coupled with an overall container adoption growth from fifty-seven to sixty-five percent. Forbes also pointed out that over fifty percent of respondents to the 2020 survey expected to use more cloud in response to the global Coronavirus pandemic. [6]

As container adoption expands throughout the cloud, concerns surrounding the detection of security problems arise from both practitioners and academicians [7-11]. Alert Logic, a big data security-as-a-service company, published a 2015 report that stated "businesses using cloud environments are largely considered a 'fruit-bearing jackpot' for hackers" [12]. The speed with which containers execute is a key concern since containers can perform their function in seconds [13], as they are typically destroyed once a task is done. Therefore, data about containers must be collected while they are performing their tasks. Introspection tools such as Prometheus [14] and Datadog [15] have been created to gather this data as an orchestrated containerized environment is running.

These concerns in both the cloud and with containers specifically prompts the hypothesis that introspection tools can be expanded to gather forensically useful data from larger, orchestrated containerized workloads. Subsidiary questions identified as part of this research are as follows:

1. Which levels of an orchestrated containerized system does an introspection tool have access?
2. How can data be saved away from the running system?
3. How can this data be visualized for ease of examination?

The research contribution of this paper is an analysis of the data that an introspection tool can gather in a multi-server orchestrated containerized environment running a variety of machine learning workloads. The paper is structured as follows. Section 2 discusses the research surrounding the intersection

URI: https://hdl.handle.net/10125/71457
978-0-9981331-4-0

of cloud computing, big data, and container orchestration software. Section 3 presents the experimental methodology and design. Section 4 examines the results of a series of experiments designed to determine what data is captured from an introspection tool from a series of containerized workloads. Section 5 draws conclusions and presents future work.

2. Related work

The National Institute of Standards and Technology [16] (NIST) lists five essential characteristics of cloud computing: on-demand self-service, broad network access, resource pooling, rapid elasticity or expansion, and measured service. NIST also describes four deployment models: private, community, public and hybrid. These definitions have standardized cloud computing nomenclature since their publication in 2009.

Big data and the cloud have become intertwined as the two technologies have matured. Yang et. al [17] pointed out the challenges and opportunities of these two ideas. After defining several sources of big data such as the Internet of Things and business, the researchers point out how cloud computing ideas can be leveraged to help provide insight into these two evolving realms through data transmission and management as well as analysis. Some of the challenges that are underscored throughout the survey are scalability and quality of service on large-scale big data jobs. Finally, the paper concludes with several ideas for a research agenda focused on risk management, big data mining, and interdisciplinary collaboration to solve pressing issues.

One of the major players in container orchestration in a virtual is Kubernetes. Burns et. al [18] wrote about the development arc surrounding Borg [19], Omega [20], and Kubernetes. The authors were integral to the various container orchestration platforms at Google. Their 2016 work describes the history of container orchestration at Google, which started with Borg. Omega was their second-generation orchestration system within their closed-source data center operations and incorporated lessons learned from Borg. The paper also describes what Kubernetes, their latest generation container orchestrator that was released to the public, obtained from Google's internal development efforts. One key point stressed by Burns et. al [18] is how containers have lightened the OS load across the fleet of Google machines and permitted data centers to be converted into application focused processing engines.

Where Yang et. al and Burns et. al were working on systems to unite overarching concepts in the virtualization world, Casalicchio and Percibali [21] focused specifically on virtual containers. They sought to determine if tools built for a non-virtual environment collected the same information as tools built to focus on the cloud. The researchers tested a battery of traditional Linux metrics including iostat and mpstat. cAdvisor [22], a container specific introspection tool suite was used as a comparison platform for specific Docker [23] statistics. Both Prometheus [14] and Grafana [24] for utilized for statistics collection. Tests centered upon CPU and Disk I/O intensive workloads. They determined different tools present similar but not completely equal results.

Watts et. al [25] also examined containers, but that research was focused on detecting malware through introspection tools. The researchers introduced a known piece of malware to an Apache server container and ran a series of tests to determine what differences, if any, appeared in the metrics that the introspection tool Prometheus produced. Through a total five different experiments, nine different metrics were identified that allowed the user to identify if a container was infected or normal.

Examining performance and resource management was the focus of Medel et. al [26]. They built a two node Kubernetes cluster on a pair of servers, and tested container creation and termination time with CPU and I/O intensive workloads, with the goal of measuring the system time required to perform the tasks. There were several drawbacks to their approach, beginning with the fact their cluster was composed of only two nodes, one of which was the master node. This is a concern, as Kubernetes does not allow for scheduling of work on the master node unless explicitly configured to deployed jobs on the master node. Even with that limitation, their initial work was one of the first to use an actual Kubernetes deployment as opposed to the popular Minikube [27]. Minikube is single node Kubernetes that is meant for personal computer resources as opposed to server compute resources that are seen in clouds; as a result, comparisons done on Minikube do not necessarily reflect typical Kubernetes deployments.

Shah et. al [28] used microservices such as WordPress to show deployment patterns through a combination of Docker [23], Kubernetes [3], and the Google Cloud Platform [29]. The paper delineates how Docker can be used to make deployments faster, while Kubernetes on the Google Cloud Platform can control the scaling of a given application. They also compare Docker Swarm, a Kubernetes-like platform designed to work natively with Docker, to Kubernetes. The paper does an excellent job explaining the

deployment patterns along with the tradeoffs and strengths of the different containerized systems.

Managing a stateful application across a container orchestration platform can be a challenge. Kubernetes attempts to address this via StatefulSets [30], which is designed to augment the Kubernetes orchestration layer by introducing persistent identifiers to sets of containers. An alternative approach was presented by Netto et. al [31], who built a coordinator-as-a-service application called Koordinator to add some fault tolerance to Kubernetes. The authors built a service layer on-top of Kubernetes as opposed to augmenting the Kubernetes orchestration layer. The Koordinator layer sits behind the proxy servers that Kubernetes configures for its CoreDNS [32] protocols to create the Kubernetes virtual network, and CoreDNS itself. Traffic is routed through that service layer if there are many requests to an application made up of many containers. Koordinator adds a read on top of a write, but experimental tests showed that there was hardly any changeover. The system was tested with sixteen writers with eight thousand simultaneous requests as well as 256 readers sending eighty thousand requests. The resource consumption was also shown.

Understanding how efficient parts of the overall system are as Kubernetes diversifies will require vigilance, and Kratzke et. al [33] worked on the networking side of Kubernetes. The research had a series of test cases ranging from a non-virtualized system to a fully containerized system using the software-defined network Weavenet [34], a popular networking framework for Kubernetes. The benchmark mapped pings between hosts, and then created a series of line graphs which compared the non-virtual system with the fully containerized one. Their research admitted that the tool for benchmarking was limited, but it was able to augment the classical iperf [35], uperf [36] by extending their usage into Kubernetes.

A major question surrounding large virtual containerized platforms is scaling the necessary monitoring tools without an extreme performance cost. Stelly et al. [37] deal with this issue via the containerization of the digital forensics process with their SCARF toolkit. They focused on scalability across large platforms using Docker Swarm, and attempted to demonstrate that high throughput to empower scalability. The group ran tests on both a legacy cluster, and a cluster with cutting edge hardware and found that several of the components of the SCARF system, such as Yahoo's OpenNSFW network [38], had large throughput gains when comparing the two systems, and could potential scale into the big data realm.

Containerization has expanded from purely computational researchers into the world writ large. One of the more interesting use-cases fuses bioinformatics, which has already been heavily involved in using cloud compute, such as Agapito et. al's [39] simulation of vessel reconstruction, with Kubernetes. Moreno et. al [40] combined Kubernetes with Galaxy developed by Afgan et. al [41] to containerize the framework to scale bioinformatics workloads into the cloud. They manipulate the workflow through a Helm [42] chart in order to allow for configuration ease. While the paper itself is short, Monero et. al provide links to both the code as well as robust documentation for configuring the product.

Containerization research has focused on solving specific problems with a specific component within an orchestrated container system. This research struggles outside of Minikube which obfuscates many core functionalities of Kubernetes in favor of ease of use. There has been minimal investigation of distributed container processing utilizing cutting edge tools in a forensic context to examine how various big data workloads are processed throughout a technology stack.

3. Methodology

This research investigates building a data pipeline in a cluster setup with an eye toward forensic analysis. The data can be used for event reconstruction across multiple servers, or as an early warning of problems across a cluster. The research is classified as an exploratory study according to Oates since it is an attempt to understand the overall research problem [43]. It expands the framework proposed in Watts et. al [25] to collect data through the stack, rather than focusing on a single container.

3.1. Experimental testing environment

The experiment is conducted on three Dell R440 1U servers. Each server has one terabyte of storage, and 168 gigabytes of RAM. The master server has a pair of Intel Xeon processors that provides forty-eight processing cores; the two slave servers have sixteen cores apiece. All three servers use the CentOS 7 [44] operating system. These servers support the Hadoop Distributed File System (HDFS) [45], Yet Another Resource Negotiator (YARN) [35], Kubernetes [3], Apache Spark [2], HiBench [46], Prometheus [14], Helm [42], and Docker [23]. An additional virtual machine was provisioned on a fourth server, which served to store the data collected. This virtual machine has eight gigabytes of RAM, eighty gigabytes of storage, and four cores and is used to house InfluxDB

[47], and Chronograf [48]. InfluxDB is used to store the data collected during the experiments and Chronograf is used to create some of the visualizations.

The Hadoop Distributed File System [45] is the storage portion of the popular Apache Hadoop platform. It is open source, and allows networked servers to share storage between them. In the experiments that will be conducted, the work and data storage will be shared between the single master and the two data nodes, which matches the configuration used in other distributed systems [49-52].

Yet Another Resource Negotiator [35] is another part of the Apache Hadoop platform, but, where HDFS focuses on storage, YARN focuses on providing compute resources for data stored on HDFS. YARN will be used to oversee and allocate computational resources for the scripts that prepare input data for the types of computational loads the experiment runs through the CentOS [44] command line interface.

Kubernetes [3] is an open-source container orchestration software that came out of Google. The platform breaks orchestration large processing jobs into a variety of layers, and the various layers allow for expansive data gathering. The open source introspection tool Prometheus [14] works through a series of targets and configuration files and that functionality can be leveraged to empower the multiple level data gathering that this experiment seeks to generate. The package manager Helm [42], a package manager for Kubernetes similar to Linux's APT, is used to build and customize Prometheus for the Kubernetes orchestrator; it enables Prometheus to utilize the endpoints that each container exposes through the Kubernetes APIs.

Docker [23] is open source container software that runs on top of multiple host operating systems. Kubernetes interfaces with Docker to schedule jobs and distribute them across a containerized environment.

YARN and Apache Spark [2] are interrelated, but YARN originated as a batch processing engine, while Spark was an in-memory analytics engine. A containerized version of Apache Spark is what actually runs the HiBench benchmarks once they have been generated using internal YARN scripting.

HiBench [46] is an Intel developed project meant to allow for a variety of computational loads to be measured on Spark clusters. It provides the initial input data throughout these experiments, and runs four different types: TeraSort [53], WordCount [54], Singular Value Decomposition [55], and Random Forest [56]. Hadoop's TeraGen, RandomTestWriter, RandomForestDataGenerator and SVDDataGenerator provide the input data through HiBench.

InfluxDB is a time-series database that takes advantage of the Prometheus HTTP API in order to permanently store each benchmarking test in its own database for easier comparisons. Putting InfluxDB in its own virtual environment also provides an independent data store away from the experimental system. Once the data is pulled from the experiments, Chronograf is used to explore the data through a series of visualizations. Figure 1 illustrates the experimental tech stack in which data flows omnidirectionally unless otherwise indicated.

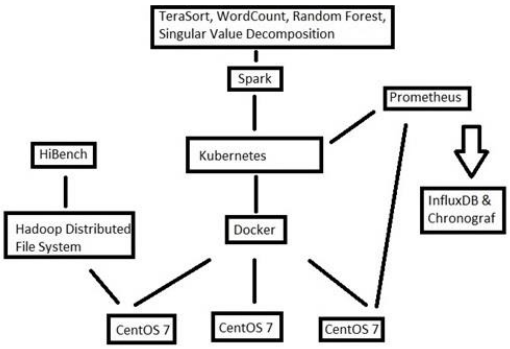

Figure 1. Experimental stack

3.2. Experimental methodology

The experiment itself combines all of the disparate elements together to build twenty different databases. Each database represents one test. There were four separate benchmarking workloads: TeraSort, WordCount, Singular Value Decomposition (SVD), and Random Forest (RF) that were each run five times. Prometheus was routed to a different database each time through the Helm package manager, and the Spark job was reformulated as necessary to go between benchmarking workloads.

Once the baseline system is built and configured so that everything is properly connected, the Prometheus Helm chart is modified between each experimental task; this modification ensures the data for an experiment is written to the proper database. In order to update Prometheus, Helm's stable/Prometheus-operator chart pull went through several iterations. First, one of the Helm configuration files, promop.values were updated to send the Prometheus UI to a nodePort, as opposed to a ClusterIP, and give it a port. The experiment used 32322 for ease for use, but any high port will function. The Prometheus UI will allow for data to be spot-checked during a test, and there is a configuration tab within the UI that prints out the underlying configuration file. That configuration file shows where

the remote_write of Prometheus is routed, and is configured through the remoteWrite portion of the values file. In order to activate remote_write, add *url: http://InfluxDB-IP:8086/api/v1/prom/write?db=<DBNAME>* inside of the brackets next to remoteWrite. The DBNAME was updated between experiments to be the name of the workload being run, and numbered one to five. Loading Prometheus from this modified repository will deploy Prometheus throughout the Kubernetes cluster, and populate approximately nine hundred different metric tables.

The spark-submit queries are all variations on a theme. The Terasort query is: *"./bin/spark-submit --verbose --master k8s://https://<KubernetesMasterIP>:6443 --deploy-mode cluster --name spark-terasort --class com.intel.hibench.sparkbench.micro.ScalaTeraSort --conf spark.kubernetes.container.image=<repo><tag> --conf spark.kubernetes.driver.pod.name=spark-terasort --conf spark.kubernetes.authenticate.driver.serviceAccountName=<username> --conf spark.executor.memory=12g --conf spark.executor.memoryoverhead=16g local:///opt/spark/bin/sparkbench/assembly/target/sparkbench-assembly-7.1-SNAPSHOT-dist.jar "hdfs://<HDFS IP>:9000/HiBench/Terasort/Input" "hdfs://<HDFS IP>:9000/HiBench/Terasort/Output.""*

Each part has a specific function within the query itself. The "—verbose" was for debugging ease throughout development. It outputs a more detailed log to the screen throughout the beginning of the query. In order for Spark to utilize Kubernetes, it has to be passed a Kubernetes IP:port combination, as well as a name for any kubectl commands. The container image is what allows Spark to run on Kubernetes, and the various properties beyond that allow for tweaking. The final two HDFS lines are the parameters of the TeraSort function.

The RF spark-submit is: *./bin/spark-submit --verbose --master k8s://https://<KubernetesMasterIP> --deploy-mode cluster --name spark-RF --class com.intel.hibench.sparkbench.ml.RandomForestClassification --conf spark.kubernetes.container.image=<repo><tag> --conf spark.kubernetes.driver.pod.name=spark-rf --conf spark.kubernetes.authenticate.driver.serviceAccountName=<username> --conf spark.executor.memory=24g --conf spark.executor.memoryoverhead=32g local:///opt/spark/bin/sparkbench/assembly/target/sparkbench-assembly-7.1-SNAPSHOT-dist.jar "hdfs://<HDFS IP>:9000/HiBench/RF/Input."*

WordCount is *./bin/spark-submit --verbose --master k8s://https:// <KubernetesMasterIP> --deploy-mode cluster --name spark-wordcount --class com.intel.hibench.sparkbench.micro.ScalaWordCount --conf spark.kubernetes.container.image=<repo><tag> --conf spark.kubernetes.driver.pod.name=spark-wordcount --conf spark.kubernetes.authenticate.driver.serviceAccountName=<username> --conf spark.executor.memory=12g --conf spark.executor.memoryoverhead=16g local:///opt/spark/bin/sparkbench/assembly/target/sparkbench-assembly-7.1-SNAPSHOT-dist.jar "hdfs://<HDFS IP>:9000/HiBench/Wordcount/Input" "hdfs:// <HDFS IP>:9000/HiBench/Wordcount/Output."*

SVD is *./bin/spark-submit --verbose --master k8s://https://<KubernetesMasterIP> --deploy-mode cluster --name spark-svd --class com.intel.hibench.sparkbench.ml.SVDExample --conf spark.kubernetes.container.image=<repo><tag> --conf spark.kubernetes.driver.pod.name=spark-svd --conf spark.kubernetes.authenticate.driver.serviceAccountName=<username> --conf spark.executor.memory=24g --conf spark.executor.memoryoverhead=32g local:///opt/spark/bin/sparkbench/assembly/target/sparkbench-assembly-7.1-SNAPSHOT-dist.jar --numFeatures 2000 --numSingularValues 1500 "hdfs://<HDFS IP>:9000/HiBench/SVD/Input."*

The Prometheus Helm chart is taken down between each test, and the values updated with the new remote_write parameters. Each job is run as the only thing within the Kubernetes ecosystem beyond the protected kube-system namespace which oversees the various containers which make up Kubernetes itself.

4. Results and discussion

The results presented below are a selection from the twenty runs. The nine hundred metrics were culled down to a handful to illustrate differences between different workloads and how those differences are visualized at various points in the stack. These visualizations are the result of queries to InfluxDB through the Chronograf visualization engine that were built off of Prometheus metrics.

4.1. Memory statistics

Prometheus is able to gather memory allocation metrics at the node level of an experimental stack. The two metrics shown are a total from an individual run, the fourth TeraSort, as well as an allocation calculated every minute throughout the run of several workloads. The total shows the system was allocating memory, but it never went down since it is a total. That was consistent across every experimental test. The individual metrics showed large differences and spikes across various nodes as processing was allocated in a cluster environment. The workloads perform different things, so differing numbers across their runtimes, which themselves were also different, is an expected behavior. This is illustrated in Figures 2 and 3.

Figure 2. Terasort 4 go_memstats_alloc_bytes_total

Figure 3. Terasort 4 go_memstats_alloc_bytes

Figures 4 through 6 illustrate difference workloads. As shown, the performance characteristics for TeraSort differs from Random Forest; this tells us that it is possible to infer different types of jobs on a running cluster. TeraSort requires a large amount of memory to store, and write out, compared to Random Forest. Wordcount and Singular Value Decomposition also have different memory profiles.

4.2. Node load statistics

Where memory statistics show differences across allocations in node memory, node load is primary concerned with processes that are currently running, plus the queued processes that follow along in order to complete a job. An important note is that these visualizations show the IP addresses [57] of the three nodes. 199.33.133.25 is the Kubernetes master node. 199.33.133.15 & 16 are the two slave nodes. Figures 7 though 10 are a selection of Terasorts and Random forests to show how node load changes between different experimental runs. Again, the differing behavior in the experiments show differences in processes counts, but these metrics also show potential differences between the individual experiments in terms of which servers run processes throughout runtimes. That type of information is valuable in event reconstruction due to being able to pinpoint when something went amiss during a security event.

Figure 4. Random Forest 1 go_memstats_alloc_bytes

Figure 5. Singular Value Decomposition 2 go_memstats_alloc_bytes

Figure 6. Wordcount 3 go_memstats_alloc_bytes

Figure 7. Terasort 5 node_load

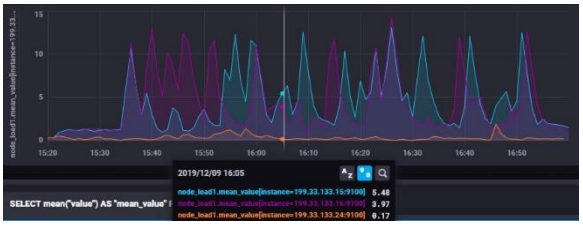

Figure 8. Tersort 2 node_load

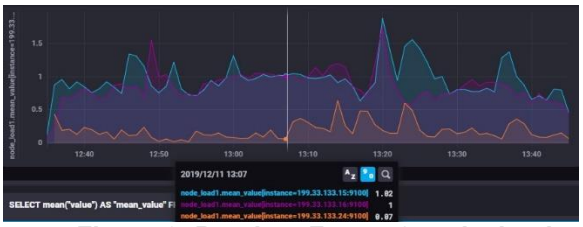

Figure 9. Random Forest 2 node_load

Figure 10. Random Forest 3 node_load

4.3. Namespace and container level filesystem statistics

Kubernetes itself is split into different levels. The main structure that Kubernetes surrounds containers with is called a namespace, and system administrators can use namespaces to spread out work between different users, or different tasks, depending on overarching policy. In a forensic context, jobs can be divided in such a way as to make pinpointing problems succinct. The namespaces hold individual containers, and Prometheus can gather both of these metrics.

4.3.1. Container_fs_usage_bytes This metric shows how the file system is utilized throughout execution of one of the experimental workload. They show how small the kube-system namespace is compared to the major running processes within the default namespace executing Spark. The development namespace holds the various parts of Prometheus. Figures 11 and 13 are namespace level metrics from Terasort 3 and Random Forest 4, and Figures 12 and 14 are container level pulls of those two experiments.

4.3.2. Container_memory_usage_bytes The other part of the system, memory, is shown in this metric. The two experiments shown are Terasort 5 and Random Forest 1. Interestingly, the major dip in Random Forest one is potentially a process changeover, or a large memory release as the classifier works through the input data. Even knowing that there was a dip has some bearing on potential event reconstruction since the data is timestamped and split amongst both namespace and containers. Figures 15 and 17 show namespace level metrics, and Figures 16 and 18 show container level metrics.

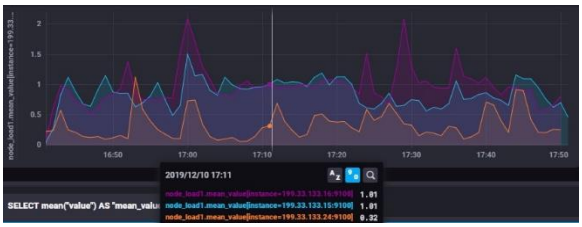

Figure 11. Namespace level Terasort 3 container_fs_usage_bytes

Figure 12. Namespace level Random Forest 4 container_fs_usage_bytes

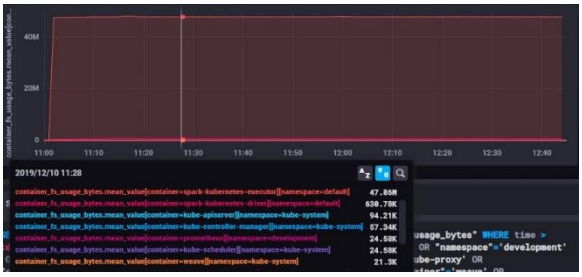

Figure 13. Container level Terasort 3 container_fs_usage_bytes

Figure 14. Container level Random Forest 4 container_fs_usage_bytes

Figure 15. Namespace level Terasort 5 container_memory_usage_bytes

Figure 16. Namespace level Random Forest 1 container_memory_usage_bytes

Figure 17. Container level Terasort 5 container_memory_usage_bytes

Figure 18. Container level Random Forest 1 container_memory_usage_bytes

5. Conclusions and future work

This research proposed three subsidiary research questions to determine whether introspection tools can be expanded across a multi-server technology stack with a container orchestrator at its heart. The first investigates which levels an introspection tool has access. The introspection tool Prometheus has access to multiple levels of a technology stack. It can pull data from the namespace and container level of Kubernetes, as well as multiple different node metrics to provide a multi-variate stream of data representing execution within the environment. The data set itself over multiple experiments contains twenty experiments of data compiled in over nine hundred separate metrics. The scaling suggests that introspection tools can be used to generate historical records for event reconstruction, or other collection surrounding processing of large amounts of data.

The second subsidiary research question dealt with constructing a one-way pipe to store data away from the orchestrated containerized experimental platform. The inclusion of InfluxDB, and the ability for Prometheus to remotely write out its metrics as they are being compiled demonstrates that it is possible to pull data out in a straightforward way, and save it outside of the running system.

The size of the data set, over nine hundred metrics, did precipitate using a visualization tool to go through them. Chronograf, built to directly interface with an InfluxDB database, was useful in answering this third and final subsidiary research question.

The answers to these subsidiary research questions show that introspection tools can expand to a large, diverse technology stack to gather relevant data for event reconstruction. No matter the workload that the orchestrated containerized system is running, Prometheus has access to relevant metrics. The metrics shown in the paper are from multiple levels in the technology stack, and show totals as well as peaks and valleys are the various pieces of the orchestrated containerized system went about the business of executing a complex, multi-server workload.

Additionally, the metrics themselves have some level of interoperability since the namespace and container level metrics look at the groupings of containers that execute a given job, as well as the individual containers themselves. That level of granularity is key to event reconstruction at the individual container level.

Future work is focused on diversification, and analysis at horizontal and vertical scale. The complex system has clear lines of demarcation between the various systems so removing one part and replacing it with a similar piece is straightforward. These comparisons have value for workload modeling, as well as studying individual parts for potential forensic analysis pitfalls. For instance, there are other container orchestrators than Kubernetes. With Docker Enterprise [58] coming for free with every copy of Window Server 2019 [59], and configured to default to Windows containers, the Azure Service Fabric [60] could be substituted. Utilizing the Service Fabric Mesh [61], which focuses on microservices on Azure, could provide a highly focused look at microservices,

and solving some of the analytical challenges inherent in that paradigm. Mirantis's Docker Enterprise Container Cloud [62] is based on the notion of clustered containers managing other clusters of containers to allow for seamless, multi-level scaling either horizontally or vertically on an ad-hoc basis.

The dataset has potential applications outside of forensics, such as resource management of large distributed systems. There were over nine hundred metrics, and targeted examinations in CPU utilization, or memory I/O, or how utilizing graphic processing unit architecture's such as Nvidia's Ampere [63] effect resource utilization are future work.

6. Acknowledgments

This work was partially supported by the National Science Foundation Grant No. CNS-1726069.

7. References

[1] Gerend, J. *Containers vs. virtual machines*. 2020; https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm.

[2] Foundation, A.S. *Apache Spark™ - Unified Analytics Engine for Big Data*. 2020; https://spark.apache.org/.

[3] Google. *What is Kubernetes?* 2018; https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.

[4] Foundation, A.S. *Apache Hadoop 2.9.1 The YARN Timeline Service v.2*. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/TimelineServiceV2.html.

[5] *2020 State of the Cloud Survey from Flexera*. 2020; https://info.flexera.com/SLO-CM-REPORT-State-of-the-Cloud-2020.

[6] Msv, J. *10 Key Takeaways From RightScale 2020 State Of The Cloud Report From Flexera*. 2020; https://www.forbes.com/sites/janakirammsv/2020/05/02/10-key-takeaways-from-rightscale-2020-state-of-the-cloud-report-from-flexera/.

[7] Ab Rahman, N.H. and K.-K.R. Choo, *A survey of information security incident handling in the cloud.* Computers & Security, 2015. **49**: p. 45-69.

[8] Agrawal, B., T. Wiktorski, and C. Rong, *Adaptive real-time anomaly detection in cloud infrastructures.* Concurrency and Computation: Practice and Experience, 2017. **29**(24).

[9] Osanaiye, O., K.-K.R. Choo, and M. Dlodlo, *Distributed denial of service (DDoS) resilience in cloud: review and conceptual cloud DDoS mitigation framework.* Journal of Network and Computer Applications, 2016. **67**: p. 147-165.

[10] Cahyani, N.D.W., N.H.A. Rahman, W.B. Glisson, and K.-K.R. Choo, *The Role of Mobile Forensics in Terrorism Investigations Involving the Use of Cloud Storage Service and Communication Apps.* Mobile Networks and Applications, 2017. **22**(2): p. 240-254.

[11] Grispos, G., W.B. Glisson, and T. Storer, *Chapter 16 - Recovering residual forensic data from smartphone interactions with cloud storage providers*, in *The Cloud Security Ecosystem*, R.K.-K.R. Choo, Editor. 2015, Syngress: Boston. p. 347-382.

[12] Palmer, D. *Hackers see cloud as 'a fruit-bearing jackpot' for cyber attacks | Computing*. 2015.

[13] Vaughan-Nichols, S.J. *What is Docker and why is it so darn popular?* 2018; https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/.

[14] Presmeg, N.C. *Prometheus - Monitoring system & time series database*. 2018; https://prometheus.io/.

[15] Datadog, *Infrastructure & Application Monitoring as a Service | Datadog*. 2015.

[16] Mell, P. and T. Grance, *The NIST definition of cloud computing.* National Institute of Standards and Technology, 2009. **53**(6): p. 50.

[17] Yang, C., Q. Huang, Z. Li, K. Liu, and F. Hu, *Big Data and cloud computing: innovation opportunities and challenges.* International Journal of Digital Earth, 2017. **10**(1): p. 13-53.

[18] Burns, B., B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, *Borg, omega, and kubernetes.* Queue, 2016. **14**(1): p. 70-93.

[19] Verma, A., L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. *Large-scale cluster management at Google with Borg*. in *Proceedings of the Tenth European Conference on Computer Systems*. 2015.

[20] Schwarzkopf, M., A. Konwinski, M. Abd-El-Malek, and J. Wilkes. *Omega: flexible, scalable schedulers for large compute clusters*. in *Proceedings of the 8th ACM European Conference on Computer Systems*. 2013.

[21] Casalicchio, E. and V. Perciballi. *Measuring docker performance: What a mess!!!* in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. 2017. ACM.

[22] Google. *google/cadvisor - Docker Hub*. 2018; https://hub.docker.com/r/google/cadvisor/.

[23] Docker. *Docker*. 2018; https://www.docker.com/.

[24] Grafana. *Grafana - The open platform for analytics and monitoring*. 2018; https://grafana.com/.

[25] Watts, T., R. Benton, W. Glisson, and J. Shropshire. *Insight from a Docker Container Introspection*. in *Proceedings of the 52nd Hawaii International Conference on System Sciences*. 2019.

[26] Medel, V., O. Rana, J.Á. Bañares, and U. Arronategui. *Modelling performance & resource management in kubernetes*. in *Proceedings of the 9th International Conference on Utility and Cloud Computing*. 2016.

[27] *Install Minikube*. 2020; https://kubernetes.io/docs/tasks/tools/install-minikube/.

[28] Shah, J. and D. Dubaria. *Building modern clouds: using docker, kubernetes & Google cloud platform*. in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. 2019. IEEE.
[29] Google. *Cloud Computing Services | Google Cloud*. 2020; https://cloud.google.com/.
[30] Foundation, A.S. *StatefulSets*. 2020; https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/.
[31] Netto, H.V., A.F. Luiz, M. Correia, L. de Oliveira Rech, and C.P. Oliveira. *Koordinator: A Service Approach for Replicating Docker Containers in Kubernetes*. in *2018 IEEE Symposium on Computers and Communications (ISCC)*. 2018. IEEE.
[32] Authors, C. *CoreDNS*. 2020; https://github.com/coredns/coredns.
[33] Kratzke, N. and P.-C. Quint. *A visualizing network benchmark for microservices*. in *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER)*. 2016.
[34] Weaveworks. *Weave Net*. 2020; https://www.weave.works.
[35] Gueant, V. *iPerf - The TCP, UDP and SCTP network bandwidth measurement tool*. 2020; https://iperf.fr/.
[36] Microsystems, S. *Uperf - A network performance tool*. 2020; http://uperf.org/.
[37] Stelly, C. and V. Roussev, *SCARF: A container-based approach to cloud-scale digital forensic processing.* Digital Investigation, 2017. **22**: p. S39-S47.
[38] *Open NSFW*. 2018; https://github.com/yahoo/open_nsfw.
[39] Agapito, G., B. Calabrese, and P.H. Guzzi. *Parallel and Cloud-Based Analysis of Omics Data: Modelling and Simulation in Medicine*. in *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. 2017. IEEE.
[40] Moreno, P., L. Pireddu, and P. Roger, *Galaxy-Kubernetes integration: scaling bioinformatics workflows in the cloud.* BioRxiv, 2018: p. 488643.
[41] Afgan, E., D. Baker, and B. Batut, *The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update.* Nucleic Acids Research, 2018. **46**(W1): p. W537-W544.
[42] Foundation, C.N.C. *Helm*. 2020; https://helm.sh/.
[43] Oates, B.J., *Researching information systems and computing*. 2005: Sage.
[44] Project, T.C. *About CentOS*. 2020; https://www.centos.org/about/.
[45] Foundation, A.S. *HDFS Architecture Guide*. 2020; https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.
[46] Intel. *Intel Bigdata -HiBench*. 2020; https://github.com/Intel-bigdata/HiBench.
[47] InfluxData. *InfluxDB 1.X: Open Source Time Series Platform | InfluxData*. 2020; https://www.influxdata.com/time-series-platform/.
[48] InfluxData. *Chronograf: Complete Dashboarding Solution for InfluxDB | InfluxData*. 2020; https://www.influxdata.com/time-series-platform/chronograf/.
[49] Vrancic, A., *Synchronization of distributed systems*. 2006, Google Patents.
[50] Shao, G., F. Berman, and R. Wolski. *Master/slave computing on the grid*. in *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000)(Cat. No. PR00556)*. 2000. IEEE.
[51] Durillo, J.J., A.J. Nebro, F. Luna, and E. Alba. *A study of master-slave approaches to parallelize NSGA-II*. in *2008 IEEE international symposium on parallel and distributed processing*. 2008. IEEE.
[52] Marquette, B.N., M.B. Stevens, M.L. Williams, and J.D. Wilson, *Master/slave architecture for a distributed chat application in a bandwidth constrained network*. 2002, Google Patents.
[53] MapR. *TeraSort Benchmark Comparison for YARN | MapR*. 2020; https://mapr.com/whitepapers/terasort-benchmark-comparison-yarn/.
[54] Foundation, A.S. *MapReduce Tutorial*. 2020; https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html.
[55] Foundation, A.S. *SVD - Singular Value Decomposition*. 2020; https://mahout.apache.org/users/basics/svd--singular-value-decomposition.html.
[56] Wang, Y., W. Goh, L. Wong, G. Montana, and A.s.D.N. Initiative, *Random forests on Hadoop for genome-wide association studies of multivariate neuroimaging phenotypes.* BMC Bioinformatics, 2013. **14**(S16): p. S6.
[57] Griffith, E. *How to Find Your IP Address*. 2020; https://www.pcmag.com/how-to/how-to-find-your-ip-address.
[58] Mirantis. *Docker Enterprise | Mirantis*. 2020; https://www.mirantis.com/software/docker/docker-enterprise/.
[59] *Windows Server 2019 | Microsoft*. 2020; https://www.microsoft.com/en-us/windows-server.
[60] *Azure Service Fabric—Building Microservices | Microsoft Azure*. 2020; https://azure.microsoft.com/en-us/services/service-fabric/.
[61] *Overview of Azure Service Fabric Mesh - Azure Service Fabric Mesh*. 2020; https://docs.microsoft.com/en-us/azure/service-fabric-mesh/service-fabric-mesh-overview.
[62] Mirantis. *Docker Enterprise Container Cloud | Mirantis*. 2020; https://www.mirantis.com/software/docker/docker-enterprise-container-cloud/.
[63] Nvidia. *NVIDIA Ampere Architecture: The Heart of the Modern Data Center*. 2020; https://www.nvidia.com/en-us/data-center/nvidia-ampere-gpu-architecture/.